

Optimizing the quality of scalable video streams on P2P networks

Ronny Vedrilla

14. Dezember 2009

Inhaltsverzeichnis

1	Problemstellung	1
2	Stand der Forschung	1
2.1	Vorangegangene Arbeiten	1
2.2	Peer-to-Peer-Netzwerke	2
2.3	Fine-grained scalable Coding (FGS)	2
3	Modell	2
4	Effizienzmessung (<i>performance measures</i>)	4
4.1	Verschwendung (<i>waste</i>)	4
4.2	Schwankung (<i>variability</i>)	5
4.3	Glätte (<i>smoothness</i>)	5
5	Algorithmen	5
5.1	BestAllocator	5
5.2	Naive Online-Algorithmen	6
5.2.1	Same-Index	6
5.2.2	Smallest-Bin	6
5.3	Hill-building Algorithmen	7
5.3.1	Largest-Hill	7
5.3.2	Mean-Hill	7
5.3.3	Wide-Hill	8
6	Empirie	8
6.1	Experiment mit zufälligen Daten	8
6.2	Experiment mit realen Daten	8
7	Weitere Forschungsaspekte	9
8	Fazit	10

1 Problemstellung

Das Paper von Rajendran und Rubenstein behandelt Downloading-Strategien für Videos in Peer-to-Peer-Netzwerken (P2P). Diese Netzwerke erfreuen sich immer größerer Beliebtheit und die Masse an bereitgestelltem Material nimmt immer weiter zu. Da Peer-to-Peer-Netzwerke aufgrund ihres Aufbaus keine feste Downloadrate und ständig wechselnde Server bieten, müssen Algorithmen speziell für diese Architektur entwickelt werden. Die Autoren konzentrieren sich in dieser Arbeit auf das direkte Betrachten eines Videos beim Herunterladen. Es sollen nun neue Algorithmen entwickelt werden, die trotz schwankender Downloadraten eine möglichst hohe und gleichmäßige Qualität garantieren und die Verschwendung der Bandbreite minimieren. Wie die bisherige Forschung gezeigt hat, empfinden Betrachter eines Videos Schwankungen in der Qualität schlimmer als eine einheitlich schlechtere Qualität. Allerdings soll die vorhandene Bandbreite möglichst gut ausgenutzt werden, da natürlich trotzdem eine hohe Qualität wünschenswert ist. Um Videos direkt anzeigen zu lassen, sind Bandbreiten von 32-300 kbps notwendig. Zum Erscheinungszeitpunkt des Artikels im Jahr 2005 wurden diese Bandbreiten vor allem in P2P-Netzen sehr selten erreicht. Obwohl heute für den PC diese Bandbreiten verfügbar sind, können solche Algorithmen beispielsweise für Handys verwendet werden. Die Internetverbindung dieser ist heutzutage in den meisten Fällen langsamer als in herkömmlichen Netzen. Des Weiteren ist gerade wegen der wachsenden Verbreitung moderner Handys Videostreaming über das Internet von besonderem Interesse. In der hier besprochenen Arbeit werden Algorithmen für optimale Downloading-Strategien von Videos entwickelt und deren Effizienz anhand simulierter und realer Daten empirisch gezeigt. Insbesondere wird ein neuer, optimaler Offline-Algorithmus vorgestellt, der zusätzlich als Performanceschranke für ebenfalls neue Online-Algorithmen dient.

2 Stand der Forschung

2.1 Vorgegangene Arbeiten

Zum damaligen Zeitpunkt wurde die Verbesserung des Betrachtungserlebnisses von gestreamten Videos in P2P-Netzen noch nicht behandelt. Diverse Autoren untersuchten ähnliche Probleme, nur bezogen sich diese auf Client-Server-Umgebungen, hatten eine feste Anzahl der Schichten oder wichen auf eine andere Weise von dieser Arbeit ab. Systeme wie „BitTorrent“ oder „Coolstreaming“ versuchen zwar ebenfalls, den Download zu optimieren, doch werden die Downloads nicht gleichzeitig abgespielt und somit muss das

Betrachtungserlebnis für den User nicht berücksichtigt werden. Das Paper kann also als Pionierarbeit gesehen werden. Trotzdem fanden die Autoren in anderen Forschungsgebieten wichtige Vorarbeiten: Die Studie von M. Zink et al. liefert die schon angesprochenen Erkenntnisse, dass eine konstante, gegebenenfalls auch niedrigere, Qualitätsstufe deutlich angenehmer empfunden wird als starke Schwankungen. Ebenfalls reagieren Betrachter sehr negativ auf plötzliche Verschlechterungen der Qualität.

2.2 Peer-to-Peer-Netzwerke

Peer-to-Peer-Netzwerke (P2P) sind eine Menge von vernetzten Rechnern, bei der jeder Peer Daten von anderen Knoten herunterladen kann und gleichzeitig Daten zum Download bereitstellt. Die Position jedes Peers im Netzwerk ist gleichwertig und somit nicht hierarchisch wie z.B. im Client-Server-Modell. Der besondere Unterschied zu diesem klassischen Modell besteht darin, dass es keinen Server gibt, der bereitgestellt werden und über eine große Bandbreite verfügen muss. Jeder Peer ist einem anderen „ebenbürtig“ und stellt seine ungenutzten Ressourcen wie Speicherplatz und Übertragungsrate bereit. So entfallen nicht nur die Kosten für einen Server, auch das Angebot der Daten und die mögliche Bandbreite werden deutlich verbessert. Das Problem dabei ist, dass jeder Peer unberechenbar das Netz verlassen oder neu einsteigen kann. Somit können die Bandbreite und die angebotenen Daten sich über die Zeit sehr stark verändern.

2.3 Fine-grained scalable Coding (FGS)

Fine-grained Scalable Coding ist Teil der MPEG-4-Codierung und besteht aus zwei Schichten (*layer*). Die erste ist eine relativ kleine Basisschicht, die zwingend heruntergeladen werden muss. Mit dieser lässt sich das Video in schlechtester Qualität betrachten. Die zweite ist die Verbesserungsschicht, welche deutlich größer ist als die Basisschicht und in M Unterschichten aufgeteilt werden kann. Jede zusätzlich heruntergeladene Schicht erhöht die Qualität des Videos, bis alle Schichten lokal vorliegen und das Video in bester Qualität betrachtet werden kann.

3 Modell

Die Autoren erstellen ein diskretes Modell, welches als Basis für die Entwicklung der Algorithmen dient. Anhand dieses Modells wird ein Offline-Algorithmus entwickelt.

Das Video wird im positiven Teil eines 2D-Koordinatensystems dargestellt. Dabei repräsentiert die X-Achse die Zeit. Das Video wird entlang der Zeitachse in T Epochen eingeteilt. t_0 ist die Zeit, die das Video zum Vorladen

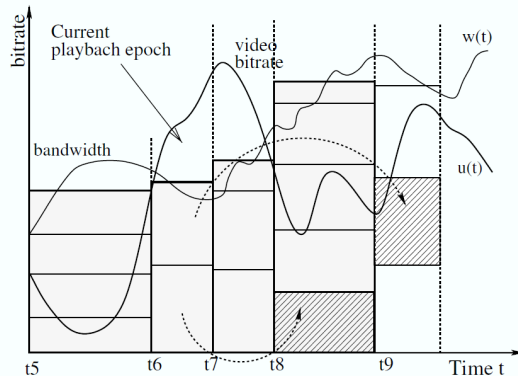


Abbildung 1: Einteilung eines Videos in Chunks

(*pre-fetching*) benötigt. Jede Epoche startet bei t_i und endet bei t_{i+1} . Auf der Y-Achse werden die Schichten des FGS eingetragen. Aus der Aufteilung des Videos in Epochen und Schichten ergeben sich Datenpakete, die Stücke (*chunks*) genannt werden. Da es sich um ein diskretes Modell handelt, sind diese Chunks atomar, das heißt nicht teilbar. Des Weiteren hat jeder Chunk per Definition die gleiche Größe. Da die Bitrate des Videos variieren kann, die Chunks aber eine fixe Größe haben, ist die Länge der Epochen variabel. In der Epoche t_i kann die verfügbare Bandbreite bereits genutzt werden, um Chunks für zukünftige Epochen herunterzuladen. Jede Epoche besitzt einen „Eimer“ (*bin*), der mit Chunks dieser Epoche befüllt werden kann. Dabei ist die Eimergröße (*bin-size*) einer jeden Epoche gleich der Anzahl der maximalen Schichten M .

W ist als ein Vektor definiert, der an der Stelle i die verfügbaren Bandbreitenslots während der i -ten Epoche darstellt. Die Anzahl der Slots zeigt an, wie viele Chunks mit der vorhandenen Bandbreite heruntergeladen werden können. Der Vektor A stellt die Zuordnung (*allocation*) dar, wie viele Chunks pro Epoche heruntergeladen und damit abgespielt werden. Eine Zuordnung A wird dann als zulässig bezeichnet, wenn eine Pre-Fetching-Strategie existiert, so dass mit der gegebenen Bandbreitenrestriktion die Zuordnung A allokiert werden kann.

Die nachfolgend vorgestellten Algorithmen benötigen einen W -Vektor als Input und liefern eine Zuordnung A als Output. Der Unterschied zwischen Off- und Online-Algorithmen liegt darin, dass Offline-Algorithmen über den gesamten Vektor W optimieren, Online-Algorithmen arbeiten hingegen iterativ und kennen erst die Bandbreite der i -ten Epoche, wenn diese beendet ist. Somit können Online-Algorithmen nicht vorausplanen, da sie nicht die Bandbreiten der Zukunft kennen.

Der Download aller noch nicht vollständig heruntergeladenen Chunks für die Periode i wird beim Beginn dieser abgebrochen, da er nicht mehr ange-

zeigt werden kann. Es ist also unvorteilhaft, Videostücke parallel herunterzuladen. Sequenzielles Herunterladen hingegen konzentriert die verfügbare Bandbreite auf einen Chunk.

Das Modell ist nur für das „normale“ Betrachten des Videos ausgelegt, lässt sich aber einfach um die Pausen- und Zurückspulfunktionen erweitern, da die Zeit, die dann zum Laden des Videos zur Verfügung steht, verlängert wird.

$u(t)$ ist die Abspielrate, also die minimale Downloadrate, die benötigt wird, um das Video in bester Qualität anzeigen zu können. Jeder Peer besitzt nur vollständige Chunks.

$w(t)$ ist als die Anzahl der freien Bandbreitenslots definiert, die in der jeweiligen Periode zur Verfügung stehen.

Damit ein Chunk effektiv heruntergeladen werden kann, sollte eine relativ hohe Bytezahl als Chunkgröße gewählt werden. Eine Empfehlung der Autoren ist die Wahl von 100KB, was in etwa 5 Sekunden eines Videos entspricht. Dies ist wichtig, damit genug Zeit bleibt, die Peers auf den Download einzustellen. Für die Planung (*scheduling*) der Downloads kann ein im Modell unteilbarer Chunk in der Praxis in Microchunks zerlegt werden, welche somit dieselbe Größe wie ein Datenpaket besitzen. Diese Microchunks können dann terminiert werden. Steigt ein Server bzw. Peer aus, muss natürlich neu geplant werden. Es kann effizienter sein, verschiedenen Servern verschiedene Microchunks zuzuweisen. Hierbei besteht allerdings die Gefahr, dass bei plötzlichem Abfall der Bandbreite viele Microchunks nicht fertiggestellt werden können.

4 Effizienzmessung (*performance measures*)

Um die Effizienz von Scheduling-Algorithmen zu bewerten, bietet es sich an, den Anteil der nicht genutzten Bandbreite als Messlatte heranzuziehen. Da jedoch ein zentraler Aspekt dieses Papers die Verbesserung der Betrachtungserfahrung ist, reicht es nicht aus, nur die Qualität der Einzelbilder als Maßstab für die Güte eines Algorithmus heranzuziehen. Basierend auf den Forschungsergebnissen von M. Zink werden zwei weitere Kennzahlen erstellt.

4.1 Verschwendung (*waste*)

Sie ist definiert als die Menge der verfügbaren, aber nicht für den Download genutzten Bandbreite.

$$w(A) = \max_{B \in F(W)} \sum_{j=1}^T b_j - \sum_{i=1}^T a_i$$

Verschwendung ist die Differenz aus bester zulässiger Allokation und der vom Algorithmus gewählten. Sie entsteht, wenn alle zukünftigen Chunks be-

reits heruntergeladen wurden, das Abspielen des Videos aber noch andauert. Dies tritt besonders auf, wenn der Algorithmus zu vorsichtig vorgegangen ist und die vorherigen Chunks auf einer höheren Qualität hätten abgespielt werden können.

4.2 Schwankung (*variability*)

Sie ist definiert als die Summe der Quadrate der Anzahl der Schichten, die nicht beim Abspielen des Videos angezeigt wurden.

$$V(A) = \sum_{i=1}^T (M - a_i)^2$$

M ist die Anzahl der Schichten, a_i die Anzahl der heruntergeladenen Chunks in der i -ten Epoche. Je stärker a_i von der Maximalanzahl der Schichten abweicht, umso größer wird die Schwankung. Ein Video auf mittlerer Qualität hat damit eine kleinere Schwankung als ein Video mit sehr schlechter und dann sehr guter Qualität trotz gleicher Anzahl fehlender Schichten.

4.3 Glätte (*smoothness*)

Die Rate, mit der sich die Abspielqualität verändert, ist wie folgt definiert:

$$s(A) = \sum_{i=2}^T |a_{i-1} - a_i|$$

Es wird der Betrag der Differenz aus der Allokation der letzten Epoche mit der aktuellen gebildet. So werden Schwankungen von einer Periode zur nächsten gemessen. Es kommt dem menschlichen Empfinden von einem „weichen“ Ablauf am nächsten.

5 Algorithmen

5.1 BestAllocator

Die Autoren stellen einen Offline-Algorithmus vor, der für eine gegebene Bandbreite eine optimale Allokation liefert. Dieser optimiert die Zuordnung, indem er den gesamten Vektor W betrachtet und rückwärts die Chunks zuteilt. Er iteriert über alle freien Bandbreitenslots jeder Epoche. Dabei werden der j -ten Epoche nur Chunks der Epochen $i > j$ zum Download zugeordnet. Gibt es mehrere Epochen i , der ein Chunk zugeordnet werden kann, so wird diejenige ausgewählt, welche die minimale Anzahl an zugeordneten Chunks beinhaltet. Steht nach diesem Kriterium immer noch mehr als eine Epoche zur Auswahl, so wird die späteste genommen. Dieses Vorgehen kann in

der Praxis nicht verwendet werden, da Online-Algorithmen nicht über dieses Wissen verfügen. Somit kann der *BestAllocator* als Performanceschranke verwendet werden, welche dazu dient, die Effizienz der Online-Algorithmen zu messen.

Aufgrund der Länge des Optimalitätsbeweises, der aus acht Schritten besteht, wird hier lediglich der Beweis für die Restriktionen „Verschwendung“ und „Glätte“ durchgeführt.

Verschwendung: Der Algorithmus betrachtet jeden freien Bandbreitenslot von Epoche T bis 1. Falls es in Epoche i eine nachfolgende Periode gibt, die noch nicht die maximale Anzahl an Chunks enthält, so wird dieser Slot zum Download benutzt. Nur im Fall, dass alle nachfolgenden Epochen bereits voll sind, wird der Slot freigelassen. Da aufgrund des Modells keine Chunks für die gleiche oder vorhergehende Epoche heruntergeladen werden können, folgt, dass kein Slot „verschwendet“ wird.

Glätte: Im Falle, dass ein Slot zum Download verwendet wird, wählt *BestAllocator* die leerste Epoche. Sollten mehrere gleich „leere“ Epochen existieren, wird die späteste gewählt. Offensichtlich gilt daher, dass die Anzahl der Chunks pro Epoche nicht-fallend ist: $a_0 \leq a_1 \leq \dots \leq a_T$. Somit ist die Glätte einer Allokation, die mit *BestAllocator* erstellt wurde, $a_T - a_0$. Dies ist nicht die minimal mögliche Glätte, da die Allokation $\langle 0, 0, 0, \dots, 0 \rangle$ eine niedrigere Glätte besitzt. Allerdings kann keine Allokation, die a_T Schichten besitzt, eine niedrigere Glätte haben als die von *BestAllocator* erreichte.

5.2 Naive Online-Algorithmen

Für Online-Algorithmen gibt es zwei Extremstrategien, die in den folgenden beiden naiven Algorithmen dargestellt sind.

5.2.1 Same-Index

Alle verfügbare Bandbreite wird dazu genutzt, für die nächste Periode Chunks herunterzuladen. Erst wenn diese Periode M Chunks enthält oder abgespielt wird, werden für weiter in der Zukunft liegende Perioden Videostücke heruntergeladen. Durch diese Strategie wird so gut wie keine Bandbreite verschwendet, allerdings entsteht ein Problem bei einem plötzlichen Abfall der Übertragungsrate, da bis dato keine zukünftigen Chunks heruntergeladen wurden. Dieses Verfahren neigt auch zu großen Schwankungen in der Bildqualität, da es sehr anfällig für Änderungen der Bandbreite ist.

5.2.2 Smallest-Bin

Dieser Algorithmus arbeitet sehr konservativ und lädt ein Chunk für die Epoche, welche die wenigsten Chunks enthält, herunter. So wird das Video

Schicht für Schicht heruntergeladen und man ist relativ gut gegen niedrige Übertragungsraten abgesichert. Ebenfalls erzeugt dieses Vorgehen eine gute Glätte und verursacht eine relativ geringe Schwankung bei der Qualität. Allerdings kann durch diese Strategie viel Bandbreite verschwendet werden, was nicht wünschenswert ist.

5.3 Hill-building Algorithmen

Es liegt auf der Hand, dass ein optimaler Algorithmus zwischen diesen beiden Extremstrategien liegen muss. Um dem Problem der Glätte beizukommen, führen die Autoren eine Abhangrestriktion (*downhill-slope*) ein, die beim Aufbau eines „Chunghügels“ das maximale Gefälle beschränkt. Die sogenannte C-Bedingung lautet: $b_i - b_{i+1} < C$, wobei $B = \langle b_1, \dots, b_T \rangle$ eine Allokation ist.

Diese Bedingung wird vor dem Download eines Chunk überprüft. Ansonsten könnte die Bandbreite nach dem Download abfallen und es besteht nicht mehr die Möglichkeit, die „fehlenden“ Chunks herunterzuladen, um das Gefälle zu beschränken.

5.3.1 Largest-Hill

Jede Epoche besitzt genau einen Eimer (*bin*), der mit bis zu M Chunks gefüllt werden kann. Dieser Algorithmus legt jeden Chunk in den Eimer mit dem kleinsten Index j , so dass die C-Bedingung erfüllt ist. Er versucht somit, dem nächsten Eimer möglichst viele Chunks zuzuordnen und tendiert dazu, Hügel mit dem Gefälle der Abhangrestriktion zu bauen.

5.3.2 Mean-Hill

μ ist als die durchschnittliche Anzahl an Chunks definiert, die den vergangenen Epochen zugeordnet wurden. Es spiegelt somit die durchschnittliche Bandbreite wider. Der Algorithmus versucht, alle Eimer mit mindestens μ Chunks zu füllen. Somit ist μ eine Höhenrestriktion. Er arbeitet nach folgender Regel:

Finde den Eimer mit kleinstem j , so dass auch die Abhangrestriktion erfüllt ist, wenn ein Chunk mehr in den Eimer gelegt wird.

Wenn dieser Eimer weniger als μ Chunks enthält, Download starten

Sonst Chunk für den leersten Eimer auswählen

Der Algorithmus lädt das Video somit schichtweise herunter.

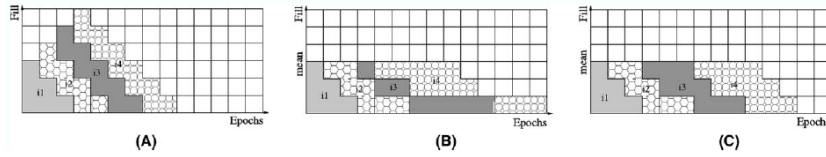


Abbildung 2: (A) Largest-Hill, (B) Mean-Hill and (C) Wide-Hill

5.3.3 Wide-Hill

Ordnet einen Chunk dem Eimer mit dem kleinsten j zu, der die Abhang- und die Höhenrestriktion erfüllt. Somit tendiert der Algorithmus dazu, einen Hügel bis zu μ aufzubauen und diesen dann immer weiter zu verbreitern.

6 Empirie

6.1 Experiment mit zufälligen Daten

Um die Güte der Algorithmen zu testen, werden die fünf Online-Algorithmen zunächst unter simulierten Bedingungen mit dem optimalen Offline-Algorithmus verglichen. Das Ziel ist die Minimierung von Verschwendung, Glätte und Schwankung. Dabei werden zwei Bandbreiten gewählt: Die erste ist eher konstant und entspricht der Bitrate des Videos. Die zweite fluktuiert immer stärker gegen Ende des Videos. Das Experiment läuft über 600 Epochen und wird 100mal durchgeführt und dann gemittelt.

Das Ergebnis ist positiv, denn die drei Hill-building-Algorithmen liegen dicht an dem optimalen Wert. Die beiden Naiven verhalten sich wie erwartet: Smallest-Bin verschwendet sehr viel Bandbreite, erzeugt allerdings ein glattes und schwankungsarmes Video. Same-Index hingegen verschwendet nahezu keine Bandbreite, schwankt aber sehr stark in der Qualität.

6.2 Experiment mit realen Daten

Um die fünf Algorithmen unter realen Bedingungen zu testen, wurden diese jeweils mit einer T1- und mit einer DSL-Leitung getestet. Es zeigte sich, dass die naiven Algorithmen wie erwartet schlecht abschnitten. Betrachtet man den DSL-Test im Detail (Abb. 3), so erkennt man, dass bei der Verschwendung alle Hill-Building-Algorithmen sehr nahe am Optimum liegen. Lediglich der Largest-Hill weicht gegen Ende etwas ab. Bei der Glätte liegen Mean- und Wide-Hill relativ nahe am Optimum, Largest-Hill ist wiederum etwas schlechter. Die Schwankung zeigt ein deutlicheres Bild. Während Wide- und Mean-Hill fast deckungsgleich sind, leistet der Largest-Hill in etwa das Gleiche wie der naive Same-Index Algorithmus. Ein ähnliches Bild

erhält man bei der Analyse des T1-Tests. Somit zeigt sich, dass in „realer“ Umgebung - im Gegensatz zu den simulierten Bandbreiten - der Largest-Hill-Algorithmus schlechter abschneidet als die beiden anderen. Diese Restriktionsalgorithmen liefern erneut fast optimale Ergebnisse.

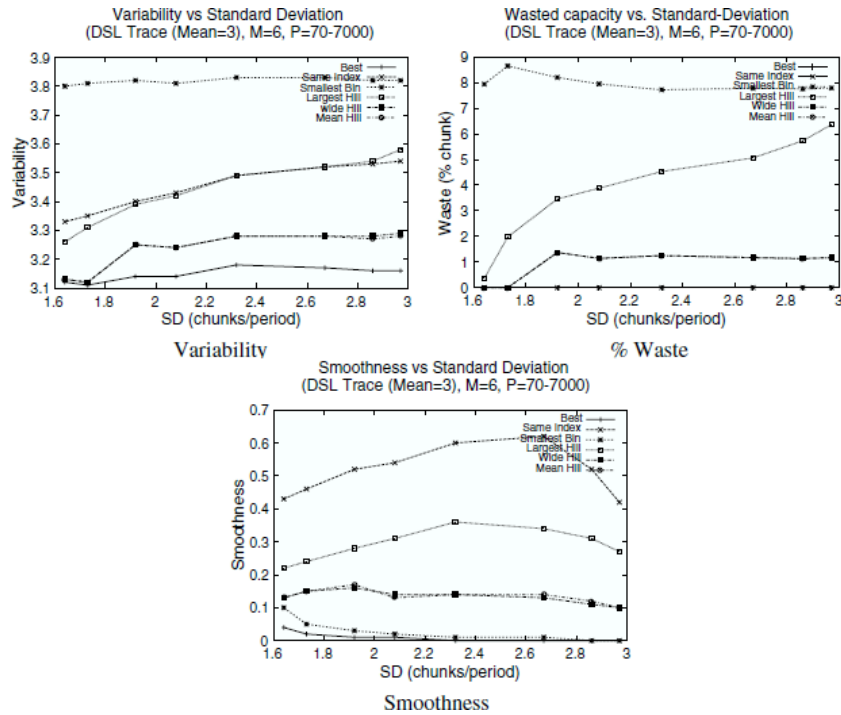


Abbildung 3: Performance der Algorithmen mit DSL-Verbindung

7 Weitere Forschungsaspekte

Die beiden Optionen des Vor- sowie Zurückspulens sollten ebenfalls betrachtet werden. Um ein Zurückspulen zu ermöglichen, müssen die heruntergeladenen Chunks lediglich für eine bestimmte Zeit vorgehalten werden. Beim Vorspulen eignen sich Algorithmen wie der Mean-Hill, da er freie Bandbreite für zukünftige Epochen nutzt und Schicht um Schicht herunterlädt, anstatt alle Anstrengungen auf die folgende Periode zu konzentrieren. Dabei kann es natürlich vorkommen, dass der Betrachter beim und direkt nach dem Vorspulen den Film in schlechterer Qualität ansehen muss, da der Algorithmus eine neue Planung erstellt. Auch wird die Zeit des „Vorladens“ in t_0 nicht weiter betrachtet. Es wäre interessant, ob und wie man diese Zeit vielleicht verkürzen könnte und welche Effekte dies auf den Gesamtprozess ausübt.

8 Fazit

Peer-to-Peer-Netze werden zum Downloaden und Betrachten von Videos immer populärer. Scalably Coded Video ist eine attraktive Lösung um von mehreren Peers Teile des Videos zu laden. Es wurden Algorithmen vorgestellt, nämlich Wide- und Mean-Hill, die sehr nahe an der bestmöglichen Leistung liegen und eine, unter den gegebenen Nebenbedingungen, gute Lösung für das Betrachten von Videos in P2P-Netzen darstellen.